Using Ada-Based Robotics to Teach Computer Science

ABSTRACT

We present an Ada-based interface to Lego Mindstorms ™, a programmable robotics kit that has attracted considerable attention in the computing community. We discuss our motivations for choosing Ada over other high-level languages, and our particular implementation over possible alternatives. Robotics and Ada combine very nicely for teaching basic computing concepts to both technical and non-technical majors, as shown with several examples.

1.0 Introduction

Computer science is an intellectually demanding discipline. Much must be taught in the classroom in a short amount of time, and much of the knowledge is foundational: newer concepts require mastery of those taught before. Accordingly, introductory computer science courses are often frustrating for both students and faculty. When asked to complete simple programming tasks that seem to accomplish little, students are frustrated at the amount of work they must put in for what seems to them little benefit. Faculty in turn are challenged by the different learning styles of their students, and by the breakneck pace of the material.

This has led computer science faculty to constantly experiment with new approaches to introductory computer science courses [HP98] [UW99]. Educators are always looking for new approaches that can a) provide a more "hands-on" approach to computing, one that is seen as both relevant and interesting by the student, and b) teach the same concepts more efficiently, presenting the essential material while requiring less time for faculty to present and students to master.

We believe that recent technological advances in the marketplace make the use of simple robotics projects a viable way to address some of these problems. We discuss our proposed system below.

2.0 Lego Mindstorms

Mindstorms is a product from Lego Corporation that enables kids (and, judging from the number of web sites, many enthusiastic adults) to construct programmable robots. The robots are built of conventional lego parts attached to a programmable Lego brick called the RCX, which contains three input ports and three output ports attached to a Hitachi H8/3292 microcontroller. A simple robot built around the RCX is shown in Figure 1. The RCX is the yellow box in the center; its input ports are connected to the black wires.

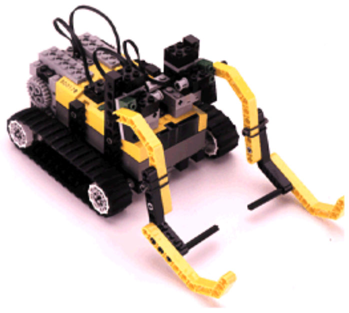

Figure 1: A Simple RCX Robot

The kit also comes with a Windows-based visual programming environment for programming the robots you build, shown in Figure 2. The resulting code is downloaded into the RCX through an infra-red transmitter (supplied with the kit) that plugs into the PC serial port.

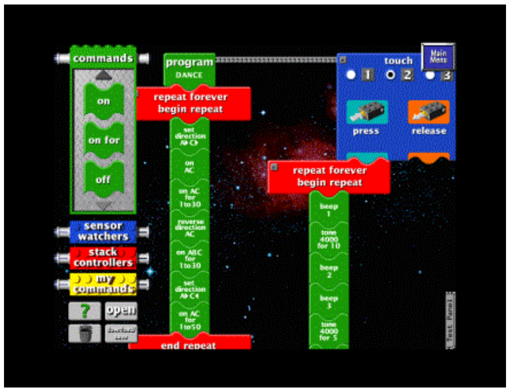

Figure 2: The RCX Programming Environment

3.0 Why a High Level Language?

While the visual programming environment provided with the RCX is perfectly adequate for its intended market, we believe it unsuited for teaching introductory computer science. Most of the programming tasks students are likely to encounter when they need to use the computer to solve a problem are not likely to be solvable with visual programming languages, nor do we see this changing in the foreseeable future. Additionally, RCX code contains no variables. This would substantially limit the kind of problems our students could solve. Finally, there's no subroutine stack in the visual environment: you can't have subroutines (called "my commands" in the RCX world, see Figure 2) call other subroutines. Again, this seems unnecessarily restrictive for computer science instruction.

4.0 Why Ada?

There are many candidates for a high level language to replace the RCX code interface. We chose Ada for several reasons. Ada has a long track record of use in introductory computer science courses [Fe99] [Le95]. Ada also embodies software engineering principles in a much more rigorous way than C or C++. Visual Basic was, we believed, too Windows-specific for the kind of general problem solving we wanted to teach, and requires users to be exposed to some rather intimidating technological baggage in order to write RCX programs. C presents well-known problems with arrays and pointers that we wished to avoid, particularly since the vast majority of students in our course will not become computer science majors. (The course we are targeting is required for all students at our institution, both technical and non-technical, regardless of their chosen field of study).

Choosing Ada also permitted us to leverage our existing Ada expertise and courseware. If we had seen evidence that other high level languages offered compelling advantages, we could have used them for the robot projects and left Ada separate for the more traditional programming efforts typical of introductory computer science classes. No such evidence was found.

5.0 Implementing the Interface

Once the language was chosen, we faced different implementation choices. One possibility was to compile Ada code directly to Hitachi bytecodes. This approach is the cleanest and most efficient, but would have required considerable effort to build an Ada/Hitachi cross-compiler. We could also have discarded the RCX firmware and compiled to a different operating environment. Again, a considerable amount of effort would have been required, so this approach was temporarily shelved, although it is a target for future projects.

Instead, we chose to leverage existing resources in the Mindstorms user community. There is a C interface to Mindstorms called Not Quite C, or NQC for short, developed by David Baum [Ba99]. Using aflex and ayacc, Ada versions of lex and yacc developed at the University of California at Irvine, we are building an Ada-to-NQC translator that takes simple Ada programs (strictly speaking, an Ada subset for Mindstorms that we will define) and converts them to their NQC equivalents. This enables us to get the Ada advantages of Ada programming for our students while at the same time saving us the effort of writing a back end for a new hardware architecture and OS.

We provide student access to the translator through a new button on AdaGIDE, the GUI interface to our Ada compiler:

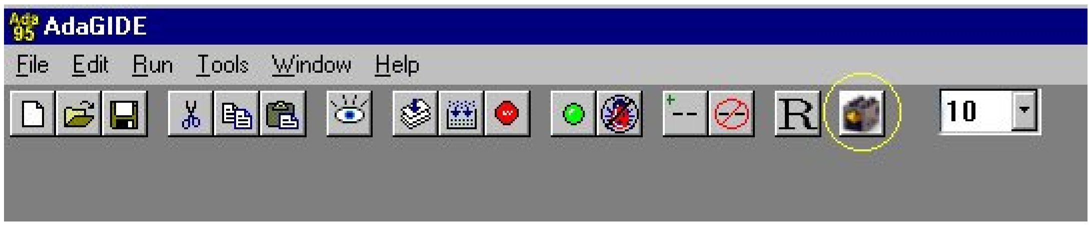

Figure 3: Accessing the Lego Interface Through AdaGIDE

If translation is successful, the NQC compiler and downloader will be invoked to send the program to the student's robot.

6.0 Examples

Since our course is a computer science course and not a robotics course, and since it is intended for both technical and non-technical majors, we focus on the programming of the robots and basic computer science constructs, not the construction of the robots themselves. Basic concepts can be illustrated with a very simple robot with two motors connected to separate wheels and output ports, and a Mindstorms touch sensor connected to a bumper and an input port.

Our goals are to introduce students to basic computing ideas, including sequential control flow, selection, iteration, input/output, arrays, graphics, procedures, and file processing. Some of these concepts easily lend themselves to robotics, others are a better suited for a more conventional paradigm.

Sequential control flow is the easiest to demonstrate. Here is a simple sequential Ada program that makes a robot move forward for two seconds, stop, turn right, go forward for one second, and stop:

```
with Lego_Package;
use Lego_Package;
procedure Sequential_Robot_Demo is
begin
    Motors_Onfwd(Outputs => Output_A +
Output_B);
    --can use additive notation to combine ports
    Wait(Time => 20);
    --Time intervals are in tenths of a second
    Motors_Off(Outputs => Output_A + Output_B);
    Wait(Time => 10);
    Motors_Onfwd(Outputs => Output_A);
    --moving left wheel causes turn to right
    Wait(Time => 10);
    Motors_Off(Outputs => Output_A);
end Sequential_Robot_Demo;
```

Selection is not too much harder, and dramatically shows students the power of decision making in a computer by giving a device the ability to react to its environment. The code below shows how to make a robot react if it runs into something by backing up, turning right for one second, and continuing:

```
if Sensor_1 = 1 then
--contact with the bumper activates a touch
sensor
    Motors_OnRev(Outputs => Output_A +
Output_B);
    --back up robot
    Wait(Time => 10);
    --for one second
    Motors_Off(Outputs => Output_A);
    --start the turn
    Wait(Time => 10);
    Motors_OnFwd(Outputs => Output_A +
Output_B);
    --and go forward again
end if;
```

Iteration can be shown with simple loop constructs. Here is a program fragment that causes the robot to play successively higher tones (the RCX unit comes with a built-in speaker):

```
          for I in 1..10 loop
              Play_Tone(Freq => I * 100, Duration => 10);
              Wait(Time => 10);
          end loop;
```

The advantages of procedures for repetitive tasks can be shown in a number of ways. Here's a procedure that makes the robot turn back and forth a given number of times:

```
          procedure Shake(Times : in Integer) is

begin
    for i in 1..Times loop
        Motors_OnFwd(Outputs => Output_A);
        Motors_OnRev(Outputs => Output_B);
        Wait(Time => 10);
        Motors_OnRev(Outputs => Output_A);
        Motors_OnFwd(Outputs => Output_B);
        Wait(Time => 10);
    end loop;
end Shake;
```

Although NQC and the RCX support additional features like data logging and a simple display, it is our current belief that topics that might use those features (like file processing, I/O, and graphics) do not map particularly well to simple robotics projects. At this point, our plans are to teach these topics using conventional programming exercises.

7.0 Conclusions and Future Work

We have only scratched the surface of the possibilities that Mindstorms presents for computer science education. Much of what is described here is dictated by our target audience of both technical and non-technical majors. If similar approaches are taken for the standard introductory course for computer science majors, considerably more sophisticated tasks can be attempted. Institutions that want to try something similar, however, need to be aware of important logistical issues, including inventory control for dozens or perhaps hundreds of Mindstorms kits.

Nonetheless, future work includes incorporating Mindstorms into our upper level programming courses, where we can develop more advanced programming challenges and demonstrate more sophisticated programming concepts like real time processing. We also are developing a simulator for a simple robot and the Ada interface, so that students can work on their robot programming projects outside the laboratory. The NQC and RCX firmware still impose limitations on how subroutining and other coding is done, so it would be useful to compile to a different operating environment like Markus Noga's LegOS [No99]. Finally, we intend to perform controlled studies to evaluate the effectiveness of using robots in promoting student mastery of basic computer science concepts.

This work was funded by a grant from the Institute for Information Technology Applications, whose support is gratefully acknowledged.

8.0 References

[Ba99] Baum, The NQC web site
[Fe99] Feldman, Ada as a Foundation Programming Language, Fall 1999
[HP98] N. Herrmann and J. Popyack, "Creating an Authentic Learning Experience in Introductory
Programming Courses", *ACM SIGCSE Bulletin,* Vol 27, No 1, pp 199-203.
[Kn99] Knudsen, The Unofficial Guide to LEGO MINDSTORMS Robots , O'Reilly & Associates, ISBN: 1565926927, buy it online here
[Le95] Levy, Computer Language Usage In CS 1: Survey Results, September 1995 SIGCSE
Bulletin, Volume 27, Number 3
[No99] Noga, The LegOS Operating System web site
[UW99] M. Urban-Lurain and D. Weinshank, "I Do and I Understand: Mastery Model Learning for
a Large Non-Major Course", *Proceedings of the 30th SIGCSE Technical Symposium on Computer Science Education,* New Orleans, LA, 1999, pp 150-154.